Fault-Tolerant Electronic Systems

There's no escaping it: sooner or later, all electronic systems fail. As a designer of embedded systems, how can you prepare for system failures? Is there a way to plan for the unforeseen glitches? George's article is a must-read for anyone with an electronic system in the pipeline.

Electronic systems, which are now more sophisticated than ever, provide us with functions and systems intelligence that was undreamed of only a few years ago. As users of these marvelous devices, we have become so reliant on their flawless operation that we can no longer imagine how we could have ever functioned without them. Unfortunately, all electronic systems fail at one time or another. The question is not if, but rather when and how they will fail. When the malfunction occurs, what will be its outcome? Will it be merely the loss of the function, which may be frustrating but usually acceptable, or will the malfunction have unpredictable, potentially catastrophic results? Will it be only a nuisance, like not being able to watch the latest DVD movie, or will you be faced with a critical, perhaps catastrophic, event, potentially causing a loss of property or even a loss of life?

A good example of the future focus on fault-tolerant embedded controllers may be the "X-by-wire" automobile that's expected to appear on the market by 2005. In the new automobile, many currently mechanical functions will be performed by electronics (e.g., steering and braking). Can you imagine having to do the infamous three-finger salute at 60 mph while the steering has decided to have a mind of its own?

Electronic systems failures can take several forms. The most desirable, of course, is fail-operational, where the system doesn't as much as hiccup and continues working as if nothing has happened. Unfortunately, for some applications, the cost of such a system may be prohibitive.

Fail-passive is the next best thing. Following a fail-passive failure, the system output assumes some predetermined desirable state, typically a power disconnect. Often, the system has a human in the loop, such as an aircraft pilot, and reverts to manual control or some benign state.

The third condition, fail-active, is highly undesirable. Although it can't be prevented with 100% certainty, it is usually allowed as a small probability, typically only 10^{-9}. In the fail-active condition, the actuators remain active but uncontrolled with unpredictable results.

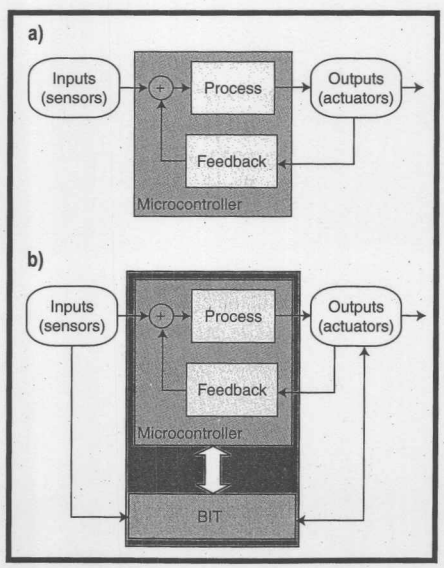

Figure 1a—*The basic system is comprised of input devices, output devices, and a controller. Not all such systems require a feedback, but it gives you some security.* ***b***—*Adding a BIT gives you a system capable of fail-passive operation.*

The causes of failures can be broadly classified into two categories. The first contains failures attributable to weak design. The design may not be robust enough for the function or the operating environment. Software bugs or other design errors fall under the same category. The reasons for a weak design may be many, but the problem can be minimized through good engineering practice, which starts with properly defined and understood specification, and continues through design reviews and analyses such as fault tree analysis (FTA), failure modes and effects analysis (FMEA), reliability prediction, etc., which are performed concurrently with the design. Finally, thorough, exhaustive testing will validate the design.

Failures in the second category are unavoidable. Whether you like it or not, failures happen: a wire breaks, a component wears out, or a bolt of lightning strikes the equipment. You can't prevent them, but you can be prepared by designing the system architecture in such a way that the effects of a failure are predictable and not catastrophic. That's what this article is about. I'll show you the different ways of making your equipment fault-tolerant.

It is generally accepted that redundancy, along with proper design, is the cure for loss of functionality because of equipment faults. You must remember, however, that there may be conditions that cannot be foreseen or that no practical redundant design can handle. Typically, if the equipment is exposed to a high-intensity radiated field (HIRF) beyond

normally expected levels, input signals in all the redundant channels can be obliterated simultaneously or a mechanical failure can sever several harnesses. The goal is to design equipment that can handle unforeseen conditions gracefully. In case of an HIRF, for example, your fail-operational controller might have to go into fail-passive mode, or hold the last position until the interference disappears. You should never consider a situation when your embedded controller goes out of control to be acceptable.

LET'S START SIMPLE

In a basic embedded control system, and it makes no difference whether it's a position control or a communications terminal, you have at least one input and one output device in addition to a processing unit, typically a microcontroller with peripherals such as memory, address decoding, and glue logic (see Figure 1a). The feedback is not always necessary, although it's a good idea because it provides the means for establishing that the process does what you want it to do. Don't assume it needs to be used only in closed-loop control systems. You may use the same principle just as well in a communications terminal. For instance, by looping the transmitted data back to the terminal, you can see if what you have put on the communications lines is in fact what you had intended.

You can improve this rudimentary design by including software built-in test (BIT) routines, having the microcontroller validate and type the input data, test the condition of the input and output devices, and validate the outputs. You can see this fundamental approach in every PC. By now it has become the minimum acceptable standard for any embedded controller. Many of today's input and output devices are BIT-able, which means they provide a way for the processor to verify their health. Some have a separate line, while others define a healthy operating current range, and so on. The same can be said of some microelectronics with internal BIT and one pin dedicated to signaling its status.

The internal watchdog timer (WDT), which is pretty much standard in every microcontroller today, can, to some degree, keep an eye on the program execution. The usefulness of the internal WDT is questionable because it can be disabled by the software it is supposed to monitor, and because it resides on the same substrate with the microcontroller that it is supposed to watch. For a little extra money, I prefer using an external WDT. Many are available, and they also include circuits for monitoring a power supply.

If the program execution goes off the rail for some reason, the microcontroller will fail (hopefully) to reset the WDT, and, in turn, the WDT will attempt to reset the microcontroller and put it back on track. If the microcontroller detects through its BIT routines a problem with a peripheral device, you can have it attempt to shut down the system or go into some predetermined state. This is all very well as long as the microcontroller operates properly and the critical peripheral devices can be controlled. If not, the system can potentially go fail-active, which is the one situation you must try to avoid.

BUILT-IN TEST

The solution to the aforementioned predicament is adding BIT equipment (BITE) that's external to the processor (see Figure 1b). The BITE independently monitors the performance of the process, including the microcontroller. The operative word here is "independently." Being independent makes the BITE capable of detecting faults within the microcontroller itself. In some situations, you may have to feed it the input signals as well, and, as is shown in Figure 1b, you will want the BITE to be able to independently shut down the outputs.

The BITE performs several functions. First, during power-up, it performs the power-up BIT (P-BIT), which verifies the memory integrity and exercises a number of functions that could not be exercised during normal operation (e.g., the WDT). Make sure you don't allow the P-BIT to overwrite stored data or cause the inadvertent movement of mechanical parts that could potentially injure someone.

The P-BIT is generally allowed to take a few seconds, but you may want to have two versions of it: a full P-BIT for cold boot and a short version for warm boot, when the several seconds of delay might not be acceptable after a manual command to re-enable.

The continuous BIT (C-BIT), which runs in the background during normal operation, is the monitor that validates data, checks for its plausibility, scans peripheral devices for integrity, and so on. If the C-BIT detects a problem, it can either initiate a fault-handler routine or activate the actuator disable (or channel control) switch. Some systems also provide initiated BITs (I-BIT), which are test routines that can be initiated manually during system maintenance to test the accuracy, control range, rigging, and so on.

FAULT PROPAGATION

Maintaining the physical independence of the processor and the BITE circuits is paramount. A common error is to use devices from a multiple-device package, such as a quad op-amp, in both the process and monitor circuits. This may save some money and PCB real estate, but it is an absolute no-no. You must also ensure that a fault cannot propagate between functional blocks.

Consider the circuit diagram in Figure 2. A fault within the sensor could cause its entire 28-VDC power to

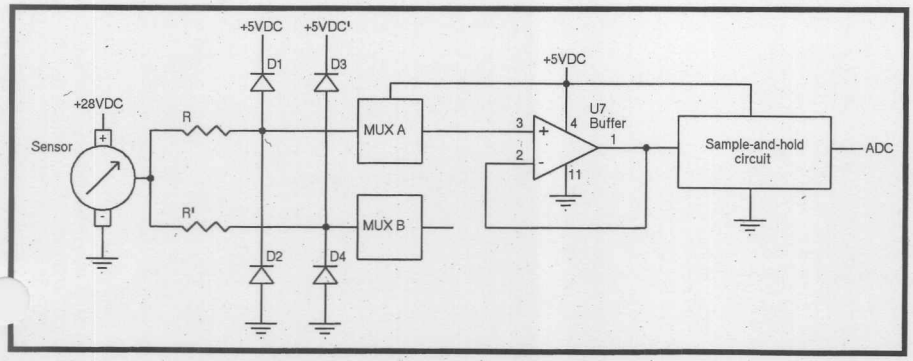

Figure 2—*Always make sure faults, whether external or internal, cannot propagate through the system and between channels.*

be placed on the multiplexer (MUX) input and destroy it. Often, this can be prevented by placing a resistor, R, in series with the source to limit the maximum current, which, together with diodes D1 and D2, keeps the input voltage to the MUX within safe limits. Many devices have the protection diodes already incorporated on the substrate, although I prefer to use discrete devices.

When using a single input source for two channels it may be tempting to use just one resistor and connect the two MUX inputs in parallel. Don't try to save the few cents. Use two resistors, R and R', as shown in Figure 2. Make sure that a fault within MUX A does not propagate to MUX B and vice versa.

Sometimes a resistor won't do the trick. Suppose a sample-and-hold circuit requiring fairly low source impedance follows the MUX and you have to use a buffer. It would be tempting to feed the buffer, usually an op-amp, from the existing ±12-V analog power. But what would happen if the buffer fails? It could feed the 12 V into the sample-and-hold circuit, which cannot sustain it, and thus cause the chain destruction of the entire channel.

Another problem could arise because of power sequencing. Not all the operating voltages come up at the same time. The situation is even worse in redundant systems that use several independent power supplies. This sequencing delay could prove catastrophic if you don't anticipate it.

It's no less important to ensure that the actuators always can be disabled. Totem pole, high-side switches are usually a good solution. It's absolutely crucial to analyze all of the possible faults and see how they will affect the rest of the system (FMEA) before the design is frozen.

How do you verify that the monitor operates properly? You can do so by having the processor test the BITE. You can inject signals out of range or otherwise disallowed to verify that the BIT picks them up. Some of this can be done only during the power-up sequence, after which it becomes a numbers game. In other words, having established that a certain feature works at power-up, such as WDT timeout, the probability of its failing during the projected operating time must be negligible. The probability of its failing followed by another fault that would, consequently, go undetected, is extremely improbable. This brings me to the topics of BIT coverage, testability, and BIT effectiveness.

BIT EFFECTIVENESS

BIT effectiveness, or fault coverage, is expressed as a percentage of all the possible faults within the device that the BIT will detect. Mathematically, the formula is as follows:

$$FD = \frac{\lambda_d}{\lambda_t} \times 100$$

where

$$\lambda_d = \sum_{i=1}^{K} \lambda_i \text{ and } \lambda_t = \sum_{h=1}^{L} \lambda_h$$

FD is the fault detection coverage as a percentage. λ_h is the failure rate for any one fault in the system. λ_i is the failure rate for the "*i*-th" identified fault. λ_d is the sum total failure rate for all of the detected faults. λ_t is the sum total failure rate for all of the faults identified in the system. Finally, note that *K* is the number of detected failures, and *L* is the number of faults identified in the system.

The fundamentals of testability and fault coverage can be found in MIL-STD-2165A, which was replaced in 1995 with MIL-HDBK-2165A. [1] The original document is hard to find, but it appears that little was changed other than the name. The document states that 80% to 95% fault coverage is considered an acceptable result. It also states the obvious fact that 100% coverage is impossible to obtain.

The document is not easy to understand, so it does not surprise me that the fault coverage number has often been torn out of context, misunderstood, and has found its way into product specifications where it doesn't make much sense. The problem, for example, is that the best practically achievable fault coverage (95%) by itself is inadequate for any system where predictable performance under adverse conditions is important. It would mean that one out of 20 failures may go by undetected, and its effects would be unpredictable. That's not acceptable under any circumstances. And when you consider highly integrated systems (e.g., a single custom IC with a multitude of passive components in the I/O lines, such as EMI filters and transient protection), the fault coverage number would be much worse because those passives are, for the most part, untestable by the BIT.

Does it mean that you may have, say, 50% undetectable failures? That's what blindly following the book and doing the math tells you. Should you care? With the exception of some extremely specific military requirements that are not directly related to safety, you really shouldn't.

To guarantee safe performance, you don't need to detect that R135 is open circuit or that low-pass EMI filter F71 is no longer providing the required attenuation at 100 MHz because the ferrite bead inside it has disintegrated. Such a depth of fault isolation could make some sense for improving maintainability and field repair in the days of discrete components.

Today, field repair of SMT assemblies is not a good idea to start with. It's more time and cost efficient to replace the entire circuit board. You need the BIT to detect when the system is not behaving properly, isolate the fault to the replaceable unit level, and initiate a predetermined corrective action, such as an actuator shutdown.

You don't need to test each component, but you should look at the system as a whole. FMEA helps identify all faults and makes sure they are all detected, albeit the majority of them indirectly. The component faults can be detected by the behavior of their functional block and the probability of their occurrence determined by the reliability prediction. Fault tree analysis is used to show that the probability of an uncontrolled failure effect (i.e., the fail-active state) is less than allowed for a given system. Your specification will customarily state that there shall be no dormant (or undetected) failures, and that the probability of a disallowed condition after a single failure does not exceed (typically) 1×10^{-9} probability.

FAIL-PASSIVE SYSTEM

Take another look at Figure 1b. Make sure the output correctly reflects the input conditions. You should create

multiple execution paths within the software and follow other good design practices. (For more information on writing software, refer to my series titled "The Joys of Writing Software," *Circuit Cellar* 121–123.)

In many instances, it is possible to design and verify the software in such a way that, if the microcontroller is working correctly, the eventuality of an incorrect output is extremely improbable. This means that in some systems the job of the external BITE design could be reduced to merely establishing that the microcontroller is working properly. The microcontroller performs the remainder of the BIT function, including the external BITE verification.

This is the idea behind the external WDT. Unfortunately, a commercially available WDT can never guarantee that the microcontroller is running properly. It doesn't take much imagination to come up with a number of conditions that can forever toggle the single WDT line, while the rest of the system is off track. All it takes is a single bit in a multimegabyte program corrupted by an external perturbation.

This problem can be solved in a number of ways. My favorite solution is for the microcontroller program to generate, within a predetermined time slot, a sequence of 4-bit tokens, which are fed into a modified WDT. In a well thought out concept, the probability of a microcontroller failure corrupting the output, while allowing the tokens to arrive on time and within the proper sequence, becomes infinitesimal.

Although many input devices are BIT-able or can be sufficiently verified merely by their data plausibility (e.g., being within a certain range or occurring in combination with others), others need to be generated by dual-redundant devices not only to verify validity, but also to guarantee that a valid input is available at all times. Devices such as dual LVDTs for displacement measurements and dual temperature or pressure sensors are commonly used in situations where accurate knowledge of the value is critical for the system's performance. Therefore, the first step in a system design must be a hazard analysis to determine the criticality of individual functions and their necessary redundancy requirements.

When processing analog inputs and outputs, such as in a closed-loop position system, verifying only microcontroller performance may not be sufficient to guarantee the required safety. Should this happen, the monitor also needs to acquire raw analog signals, as shown in Figure 1b, and process them internally. Often, this can be accomplished using the BITE FPGA plus ADCs, DACs, and table look-ups. Most often, the BIT needs to verify the output to be within a certain window only. Such single-channel, self-monitoring architecture, with the BITE capable of shutting it down when a fault is detected, can be used in a fail-passive system. It will also serve as a building block for fail-operational systems with little change.

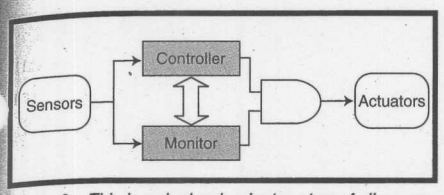

Figure 3—*This is a dual-redundant system. A disagreement between the processing and monitoring channels results in the disabling of the actuator.*

An obvious question arises: Why not build two identical, simplex, microcontroller-based channels and simply compare their outputs, as is depicted in Figure 3, instead of bothering with designing an FPGA BITE? In this dual-redundant architecture, one channel would provide the control, and the other would be the monitor. The channels exchange data, and if they both agree, the system remains operational. If they differ, the system goes passive.

This is, in fact, another common way of building fail-passive systems. However, to avoid common mode errors that could be caused by design, an internal bug in the microcontroller, a software bug, or an external condition, it is often required that you at least use dissimilar software in the two channels. It's preferable to use dissimilar hardware as well.

The cost of the development and certification of what amounts to two different controllers, usually requiring level A software (per DO-178B), is prohibitive when compared to the architecture in Figure 1b. It's easier and less costly to create and, more importantly, validate a monitor fully implemented in hardware, even with the requirement to follow DO-254. [2] Well-designed hardware can be fully testable, but software rarely is. A hardware monitor may allow you to reduce the software criticality, and the self-monitoring channel can be a useful building block for redundant, fail-operational systems at little extra development cost.

Another alternative, given here merely for completeness, is to have the dual-redundant channels fully created in analog circuitry (see Figure 3). This is an obsolete concept that provides no benefit of flexibility in today's environment.

FAIL-OPERATIONAL SYSTEMS

By employing a pair of self-monitoring controllers, you can create a fail-operational system that dual-redundancy by itself cannot. (Two channels let you determine that they do not agree, but you cannot tell which one is wrong.) You only need to slightly modify the monitors (that's also why using an FPGA is a good choice) to perform arbitration and switching between the channels. A usual alternative is to make one channel dominant and the other submissive at power-up. The dominant channel controls the process, and, while relying on its own BIT for monitoring, it also exchanges status with the submissive channel that is running in parallel. A problem detected within the dominant channel causes its shutdown and control is taken over by the submissive channel, which becomes dominant.

The microcontrollers exchange data via a serial interface. CAN bus is quite popular and reliable. But, this data exchange is noncritical; it's merely a performance enhancement. The dominant-submissive switch-over

interface between the monitors/arbitrators is accomplished by a hard connection. For obvious reasons, microcontroller communications would be suspect at this point. Besides, each channel must be fully operational with the other channel dead. To filter out noise, the switchover time is usually in milliseconds, which most mechanical systems don't even notice.

There are numerous ways to arrange the operation of the channels. One typical method is to have both channels actively control with their outputs summed up. This can be done in many ways. For example, when the actuator is a current-driven torque motor (electrohydraulic servo valve, or EHSV), its two coils may be used in parallel, each fed by one channel. In case one of the channels shuts down, you can either enter into a reduced authority mode by driving the torque motor at only 50% or raise the gain of the remaining operational channel to deliver 100%. Your choice depends on the system characteristics and the result of the hazard analysis.

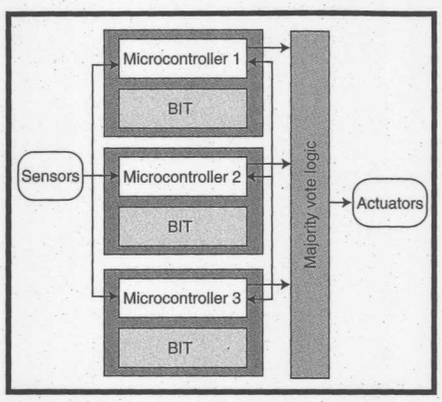

Figure 4—*This triple-redundant system uses three independent computers with majority vote.*

Another method is to have the second channel in a standby or powered-down mode and to activate it only when it's needed to take over control. This way, the spare channel does not accumulate any run-time. But, in some systems, it may take an unacceptably long time for it to come up and take over. Consequently, it is not seen too often in safety-critical systems.

When considering fail-operational architecture, don't forget the mechanical aspects of the electronic controller (i.e., the packaging design). You must look at it as if the controller has the characteristics of a dew worm: if you cut it in half, each half continues to live on its own. There must be no fault propagation from one channel to the other; communications between them must be nonessential, and absolutely no components may be shared. Normally, a metal partition inside the cabinet ensures that a fire, for instance, in one channel cannot propagate to the next. Each channel has its own harnesses and connectors. Some dual-channel aircraft systems reside in two separate cabinets, each located in a different part of the aircraft to prevent a mechanical event from wiping out the entire system.

TRIPLE REDUNDANCY

To further increase the reliability and availability of fail-operational systems, triple and even higher redundancy is used, as shown in Figure 4.

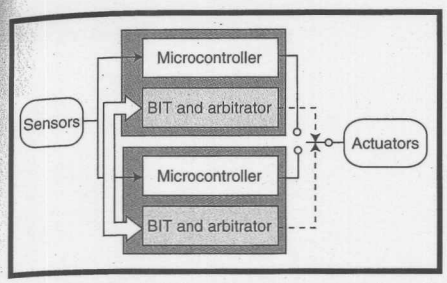

Figure 5—*The dual-channel system uses a self-monitoring pair to achieve fail-operational configuration.*

There are many aspects to consider. You can use ordinary, simplex channels, as in Figure 1a, or self-monitoring channels, as shown in Figure 1b. For instance, in making the decision, you will have to consider the allowable tracking tolerances. If the outputs of the three channels must track perfectly, you will have to use three channels with identical hardware and software, and you'll probably end up synchronizing them. But that introduces the higher probability of a common mode fault within the system. I cannot think of a position control where such perfect tracking would be required and the accompanying danger of common mode faults preferable to having dissimilar hardware and software. I would also feel more comfortable with the self-monitoring arrangement, especially when the cost of the FPGA monitor is negligible compared to the entire system and its criticality.

The idea is that the triple and more redundant systems control on the basis of a majority vote. Voting is accomplished in an external module; but, the processing units often exchange input and status data for "pre-voting" as well. This internal data exchange, if not properly designed, can lead to an unwanted effect, when no two outputs are the same. This is called a Byzantine Generals' Problem. When designing the data exchange and voting scheme, this probability must be considered. It can be avoided, for example, by using the MVS algorithm, which selects the output that is between the other two. The external voting module can be implemented electronically, or it can be mechanical or hydraulic. This depends on the specifics of the system.

We have considered what happens when the system experiences a single fault. Most fail-passive systems accept a 10^{-9} probability of a critical failure (i.e., the fail-active state) caused by a single fault. This represents one failure in more than 100,000 years. The probability of a dual failure is exponentially less. In real life, however, the impossible does happen. As British Prime Minister Disraeli once said, "there are lies, damned lies, and statistics." One time I experienced an "impossible" event that had a 10^{-9} probability of happening within the first minute of operation of the first production unit. Although extremely embarrassing, the subsequent design review showed no design flaw. The event hasn't repeated itself for almost 15 years.

What if you have a critical system and the statistically impossible is not impossible enough? The approach is essentially the same as for the single fault: you just increase redundancy. There are, for instance, triple-redundant flight computers in which each of the three computers is triple redundant itself, using three different processors from three different manufacturers to avoid common mode failures.

EXPECT THE UNEXPECTED

In summary, let's review the different architectures, keeping in mind that there are many combinations of those basic principles. Starting with the plain vanilla simplex controller like the one in Figure 1a, you must recognize that it is unsuitable for any system where safety and reliability may be of concern. The first failure puts it out of commission and the repercussions are anybody's guess.

A fail-passive system, composed of a self-monitoring controller, as shown in Figure 1b, or a dual-redundant configuration as in Figure 3, will satisfy many requirements. After the first failure, it will shut down or assume some other predetermined state with a probability of usually less than 10^{-9} of going fail-active. After the system deactivates, a second failure should not be a concern.

A dual self-monitoring system like the one in Figure 5 will continue to operate after the first failure; it goes fail-passive after the second failure. Triple-redundant (and higher) systems will potentially remain fail-operational until the last channel, which will be fail-passive.

Finally, when designing an embedded controller, it is extremely important to consider failures that can affect all of the channels at once. A good example is a signal corruption, which can be the result of a lightning strike, HIRF, mechanical failure, etc. Such faults differ from simple component failures in that they can and usually do affect the processing channels simultaneously. Therefore, it is important to understand how the system will behave under such circumstances and make sure that it does not become fail-active. Ensuring RF immunity and full functionality at 200 V/m is one thing you can do. But, what happens if the system is irradiated by a burst of 400 V/m? Or what if the shielding gets disconnected? You must always consider the unexpected and make sure the results are predictable.

George Novacek has 30 years of experience in circuit design and embedded controllers. He is currently the general manager of Hispano-Suiza Canada, a division of the Snecma Group, the world's leader in manufacturing propulsion and landing gear systems. You may reach him at gnovacek@nexicom.net.

REFERENCES

[1] U.S. Department of Defense, "Testability Program for Systems and Equipments," MIL-HDBK-2165, 1995.

[2] RTCA, Inc., "Design Assurance Guidance For Airborne Electronic Hardware," DO-254, 2000.

RESOURCES

G. Novacek, "A Sure Thing: Guaranteeing 99.99999% Reliability," *Circuit Cellar* 129.

———, "Designing for Reliability, Maintainability, and Safety," *Circuit Cellar* 125 and 126.